Introduction to Binary Trees

15-121 Fall 2020

Margaret Reid-Miller

Exam 2 is next Thursday, November 12

Topics:

- Writing methods for classes that implement Lists.
- Methods using Lists w/ ArrayList or LinkedLists
- Recursion – call tree, trace, implement
- Interfaces
- Stacks & Queues (implementations, using them)
- Evaluate post-fix expressions (not implementation)
- Big-O

Today

- Quiz 7 graded
- Autolab
  - solutions to homework written and labs
  - homework feedback

Today

- Introduction to Binary Trees
- Binary Tree Traversals

- We use what keyword to create a subclass?

  extends

- A subclass can have direct access to a field of an ancestor class with which visibility modifiers?

  `public` or `protected`

- Can you override a superclass constructor?

  No

- How do you call the superclass constructor?

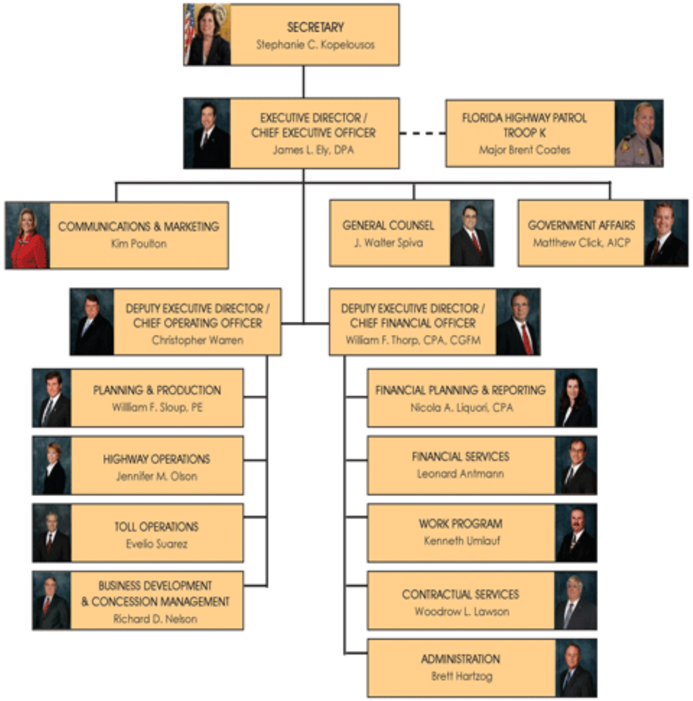
  `super()`

- Can you call it anywhere in the subclass constructor?

  No, must be the first statement

Trees

NEW YORK RANGERS
3
1
NHL WESTERN CONFERENCE
NHL EASTERN CONFERENCE



SECRETARY
Stephanie C. Kopelousos

EXECUTIVE DIRECTOR / CHIEF EXECUTIVE OFFICER
James L. Ely, DPA

FLORIDA HIGHWAY PATROL TROOP K
Major Brent Coates

COMMUNICATIONS & MARKETING
Kim Poulton

GENERAL COUNSEL
J. Walter Spiva

GOVERNMENT AFFAIRS
Matthew Click, AICP

DEPUTY EXECUTIVE DIRECTOR / CHIEF OPERATING OFFICER
Christopher Warren

DEPUTY EXECUTIVE DIRECTOR / CHIEF FINANCIAL OFFICER
William F. Thorp, CPA, CGFM

PLANNING & PRODUCTION
William F. Sloup, PE

HIGHWAY OPERATIONS
Jennifer M. Olson

TOLL OPERATIONS
Evelio Suarez

BUSINESS DEVELOPMENT & CONCESSION MANAGEMENT
Richard D. Nelson

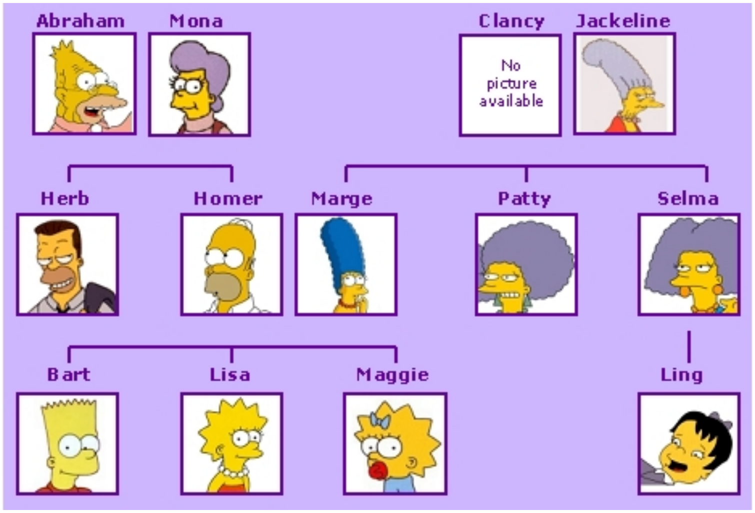
FINANCIAL PLANNING & REPORTING
Nicola A. Liquori, CPA

FINANCIAL SERVICES
Leonard Antmann

WORK PROGRAM
Kenneth Umlauf

CONTRACTUAL SERVICES
Woodrow L. Lawson

ADMINISTRATION
Brett Hartzog

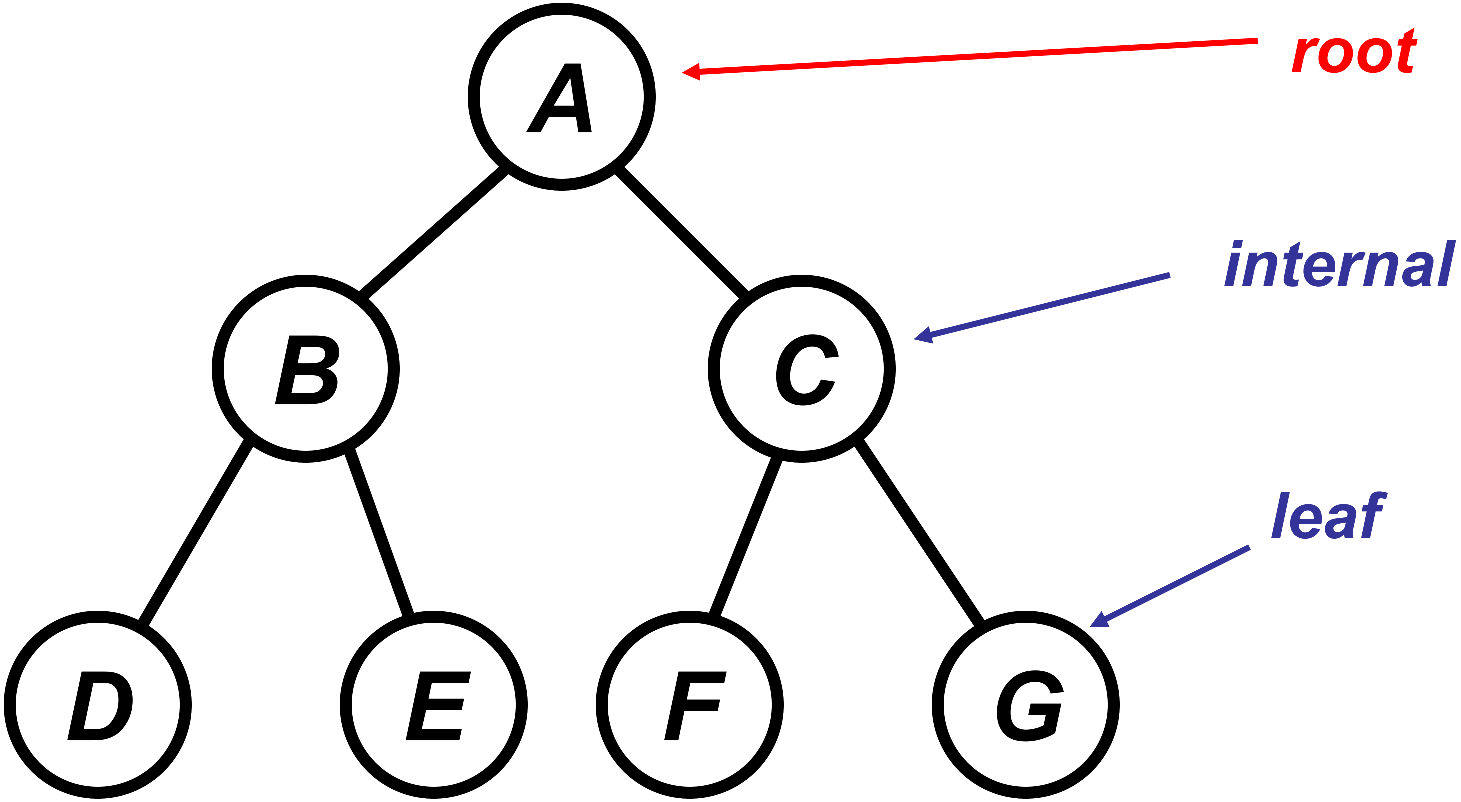


A binary tree is a nonlinear data structure

- A binary tree is either
  - empty or
  - has a root node and left- and right-subtrees that are also binary trees.

- The top node of a tree is called the root.
- Any node in a binary tree has at most 2 children.
- Any node (except the root) in a binary tree has exactly one parent node.

Tree Terminology

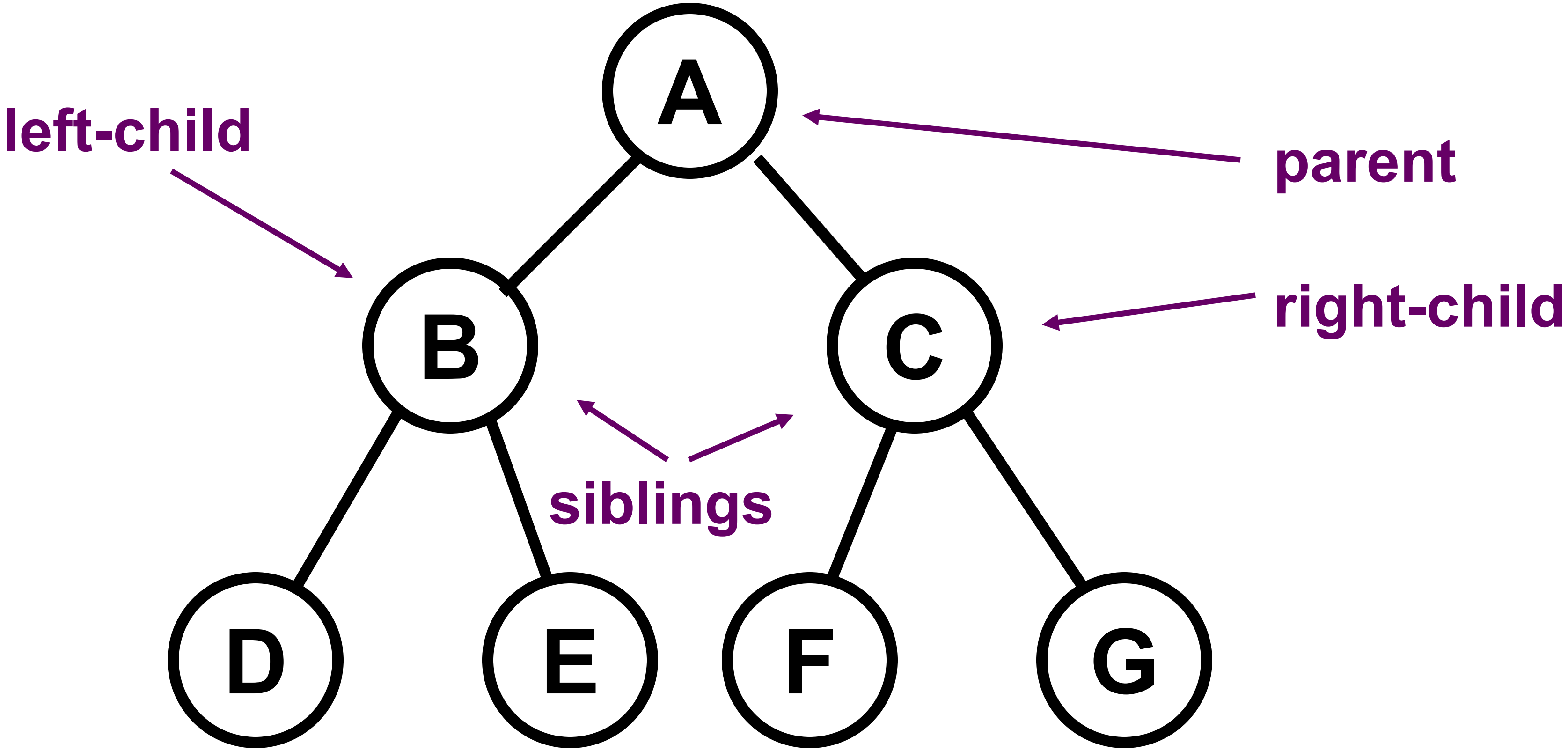


root

internal

leaf

Tree Terminology



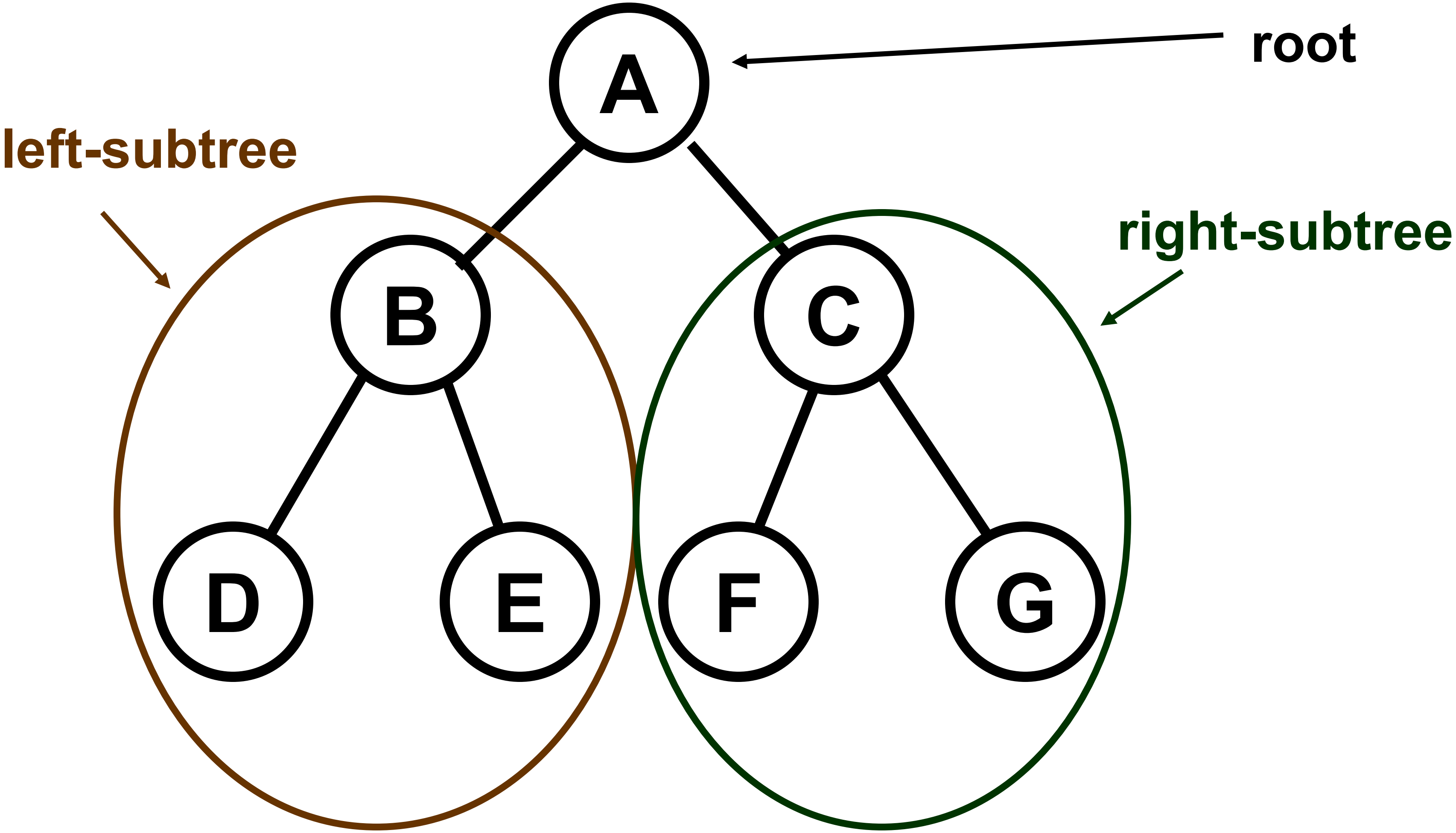
left-child

parent

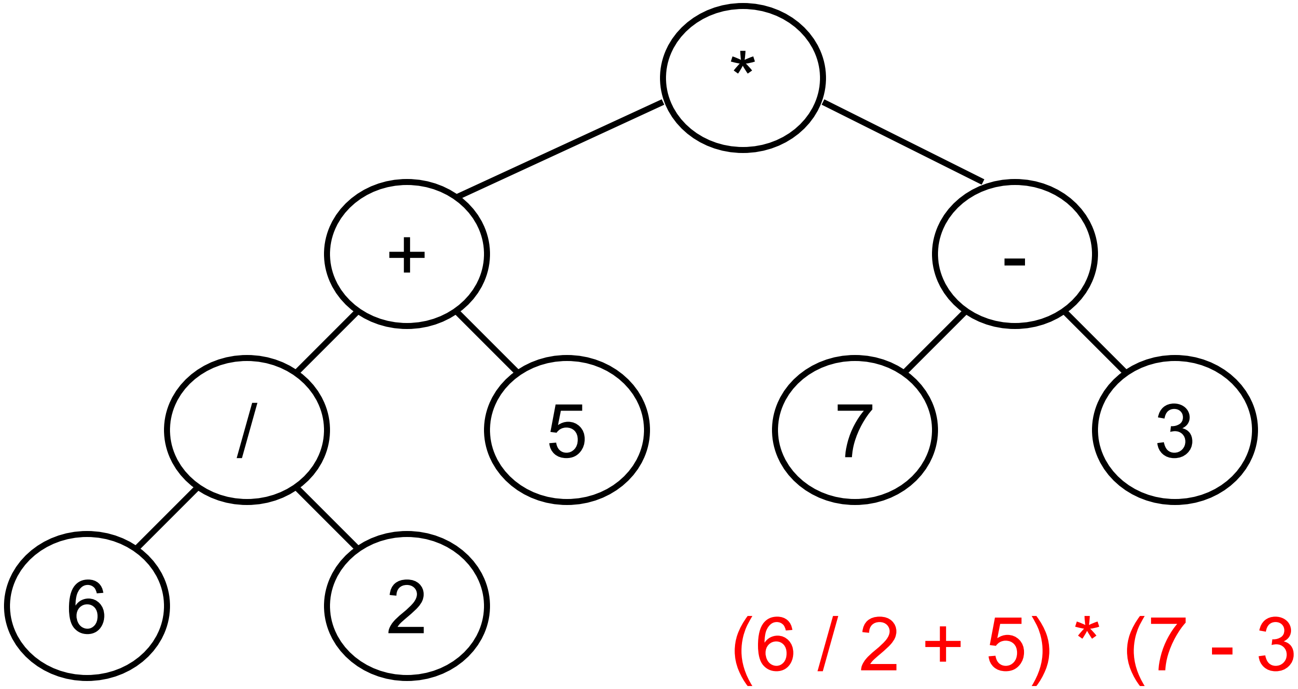
right-child

siblings

Tree Terminology



Example: Expression Trees

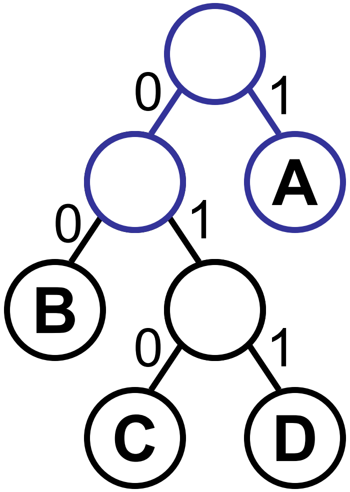


(6 / 2 + 5) * (7 - 3)

Example: Huffman Tree (data compression)

Build the Huffman tree bottom up, lowest frequencies first

	frequency
A	45%
B	30%
C	20%
D	5%



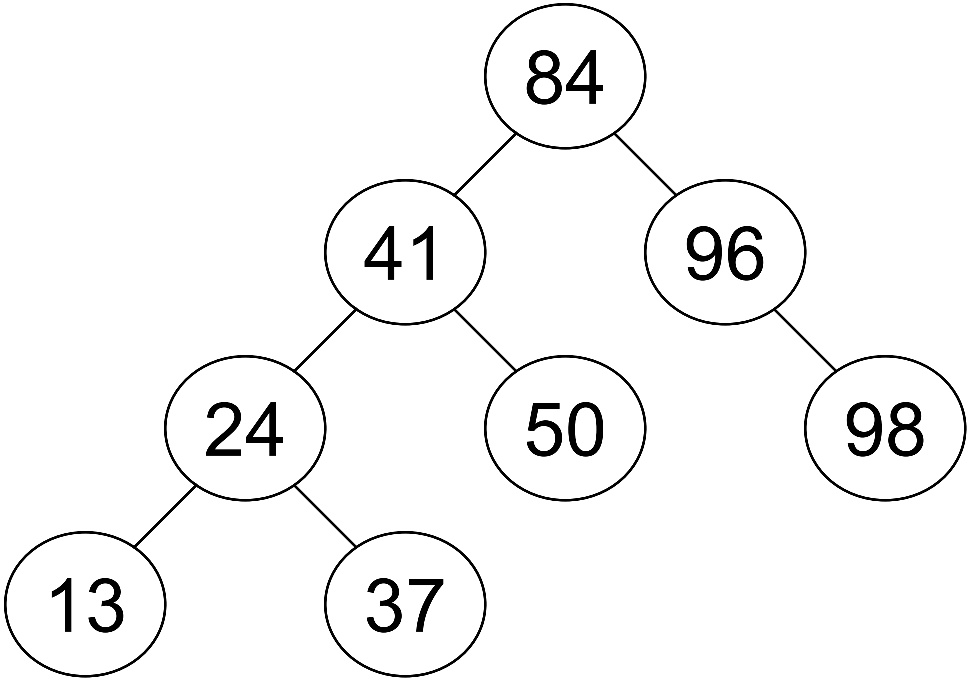
To encode: replace letter with codeword

	codeword
A	1
B	00
C	010
D	011

To decode: traverse tree
- if 0 go left,
- if 1 go right

1001010100 = ABACAB

Example: Binary Search Trees



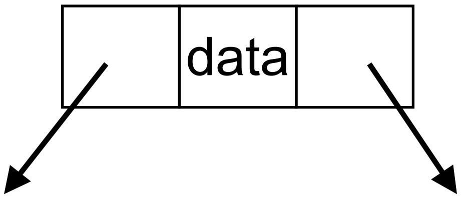
Implementing a binary tree

- Use an array to store the nodes?
  - useful for mainly complete binary trees
  (more on this soon)

- Use a variant of a linked list where each data element is stored in a node with links to the left and right children of that node.

- Instead of a head reference, we will use a root reference to the root node of the tree.

Binary Tree Node

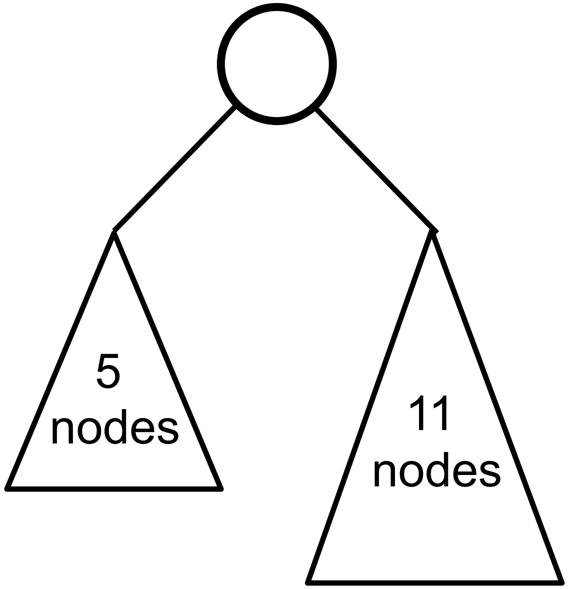


```
public class BTNode<E> {
   private E data;
   private BTNode<E> left;
   private BTNode<E> right;

   public BTNode(E d)
      { data = d; left = null; right = null; }

  public E getData() { return data; }
  public BTNode<E> getLeft() { return left; }
  public BTNode<E> getRight() { return right; }

  public void setData(E d) { data = d; }
  public void setLeft(BTNode<E> lt) { left = lt; }
  public void setRight(BTNode<E> rt) { right = rt; }
}
```

Size of a binary tree

- How many nodes are in this tree?

5 + 11 + 1 nodes



The size of a tree T is

BASE CASE

0, if T is empty

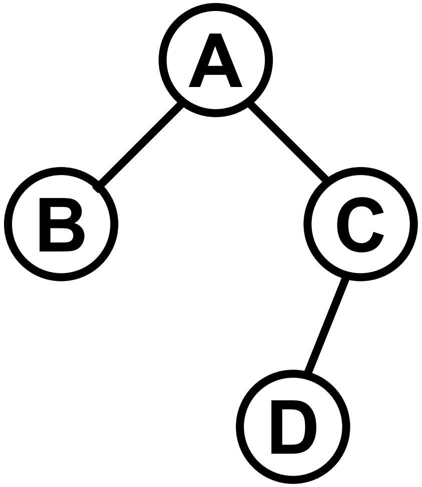
RECURSIVE CASE

1 + size of left(T)
+ size of right(T)

size() - number of nodes in t

```
public static int size(BTNode<String> t) {
   if (t == null)
      return 0;
   else
      return 1 + size(t.getLeft())
               + size(t.getRight())
}
```

t	size(t)
null	0
B	1
D	1
C	2
A	4



Maximum in a non-empty binary tree

Think recursively:

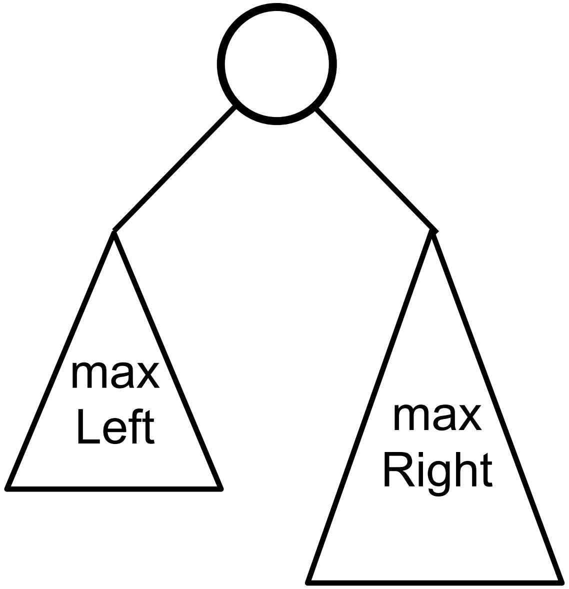
The max of a tree T is

BASE CASE

root, if T is a leaf

RECURSIVE CASE

max (root, max of left(T)
+ max of right(T))



max() – maximum in t

```
//precondition: t is not empty
//returns the maximum value in t
public static int max(BTNode<Integer> t) {

   if (t.getLeft() == null && t.getRight() == null)
      return t.getData();
   else if (t.getLeft() == null)
      return Math.max(t.getData(), max(t.getRight()));
   else if (t.getRight() == null)
      return Math.max(t.getData(), max(t.getLeft()));
   else
      return Math.max(t.getData(),
             Math.max(max(t.getLeft()),
                      max(t.getRight())));
}
```

`max()` – maximum in `t`

```
//precondition: t is not empty
//returns the maximum value in t
public static int max(BTNode<Integer> t) {
   int max = t.getData();

   if (t.getLeft() != null){
      int left = max(t.getLeft());
      if (left > max) max = left;
   }
   if (t.getRight() != null) {
      int right = max(t.getRight());
      if (right > max) max = right;
   }
   return max;
 }
```

Alternate solution

Three ways to traversing a binary tree recursively.

- **Preorder traversal**

  1. Visit the root.
  2. Preorder traversal of the left subtree.
  3. Preorder traversal of the right subtree.

- **Inorder traversal**

  1. Inorder traversal of the left subtree.
  2. Visit the root.
  3. Inorder traversal of the right subtree.

- **Postorder traversal**

  1. Postorder traversal of the left subtree.
  2. Postorder traversal of the right subtree.
  3. Visit the root.

Preorder = root, left, right

What is preorder of A's left subtree?

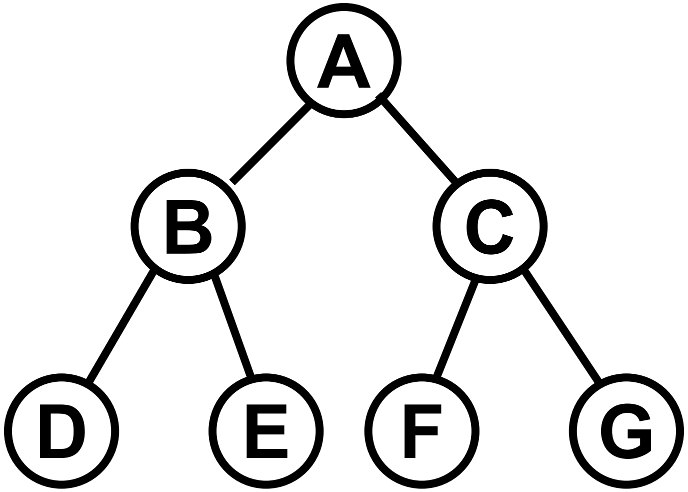
BDE

What is preorder of A's right subtree?

CFG

What is preorder of whole tree?

A BDE CFG



Return string of a Preorder Traversal

```
// Returns the elements of t as a string using
// pre-order traversal
public static String preorder(BTNode<String> t){

  String result = "";
  if (t != null) {
     result += t.getData() + " ";
     result += preorder(t.getLeft()) + " ";
     result += preorder(t.getRight()) + " ";
  }
  return result;
}
```

Binary Tree Traversals

●	PREORDER	ABDECFG
▲	INORDER	DBEAFCG
■	POSTORDER	DEBFGCA